Test Driven Development of Scientific Models

Tom Clune

SIVO (Code 610.3)

Outline

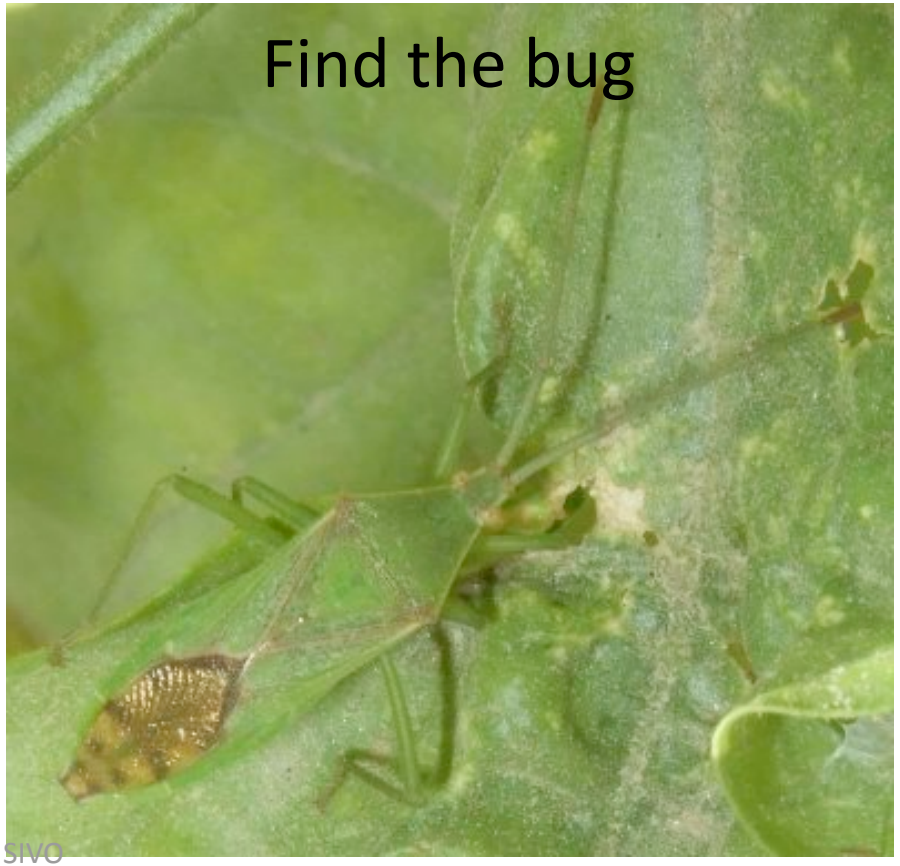
- Familiar stories
- Development
- Testing
- Test *Driven* Development
- TDD and Scientific Computing
- pFUnit – a Testing Framework for Fortran

Familiar Stories

The Marathon

The Chain Reaction

The Investigation

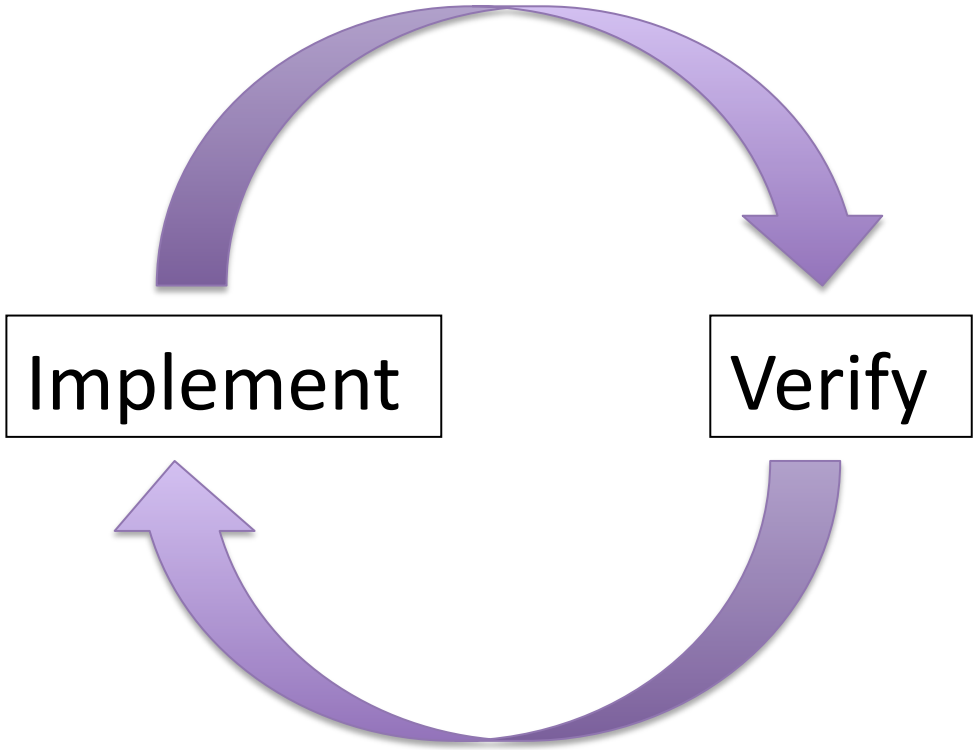


The High Wire Act

Development

The Development Cycle

- **Add code**
- **Fix bug**
- **Port**



- **Compiles**
- **Runs**
- **Plausible**
- **Correct?**

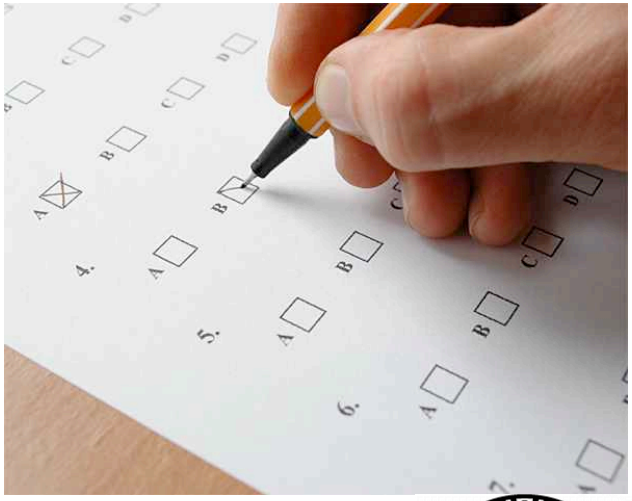
Natural Time Scales

- Design
- Implementation
- Compilation
- Batch
- Execution
- Analysis

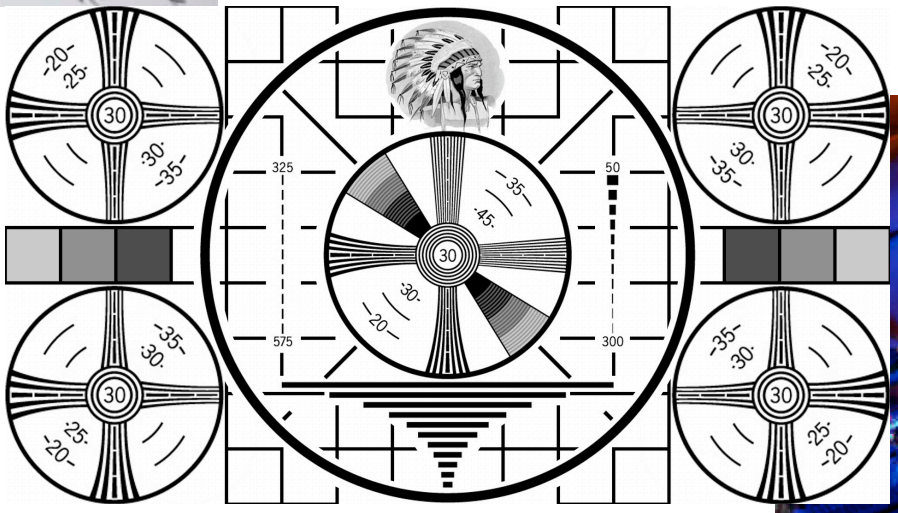
Size of Implementation Step

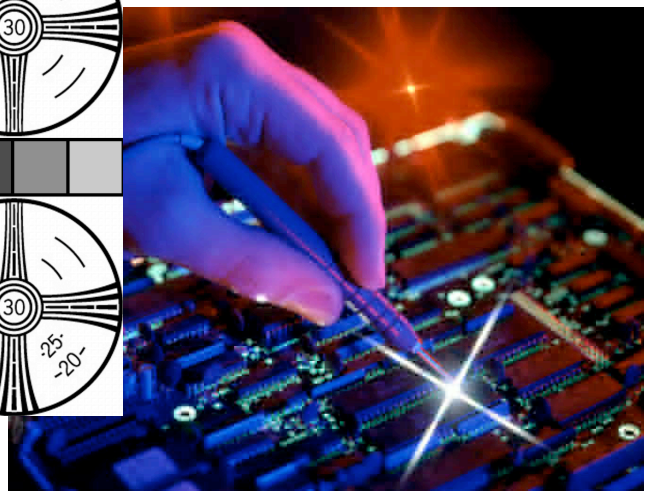
- Risk of error grows with size of change
- Size of change grows with cost of verification

- Conclusion:
  - Optimize development cycle to enable smaller changes per iteration



Testing





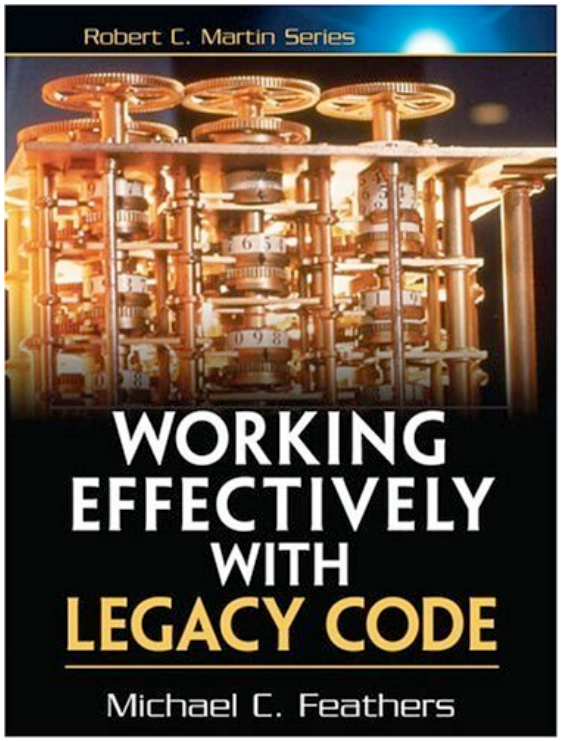
Test Harness

- A collection of tests that constrain system
- Detects unintended changes
- Localizes defects
- Improves developer *confidence*
  - Decreases risk from change

Do you write legacy code?

"The main thing that distinguishes legacy code from non-legacy code is tests, or rather a lack of tests."

– Michael Feathers
Working Effectively with Legacy Code



Lack of tests leads to fear of introducing subtle bugs and/or changing things inadvertently.

- Programming on a tightrope
- Barrier to involving pure software engineers

Excuses

- Takes too much time to write tests
  - Too expensive to *maintain* tests
- It takes too long to *run* the tests
- It is not *my* job
- "Correct" behavior is unknown

http://java.dzone.com/articles/unit-test-excuses

- James Sugrue

What is a Test?

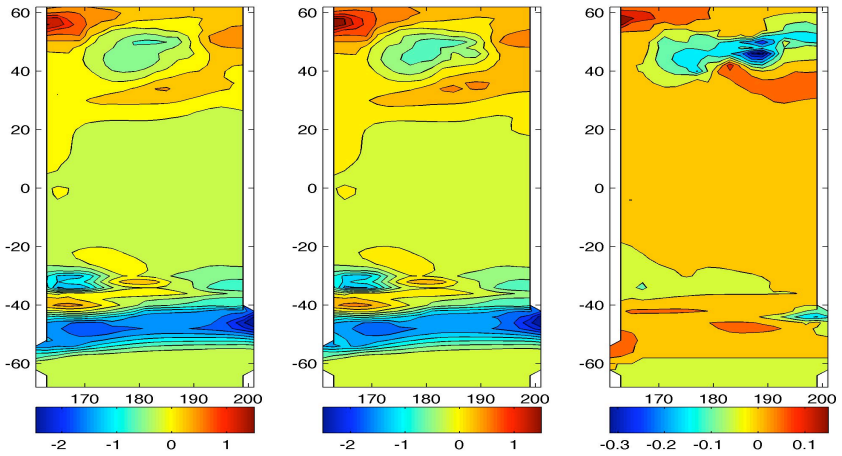
- Abort:

```
 IF (PA(I,J)+PTOP.GT.1200.)
&    call stop_model('ADVECM: Pressure diagnostic error',11)
```

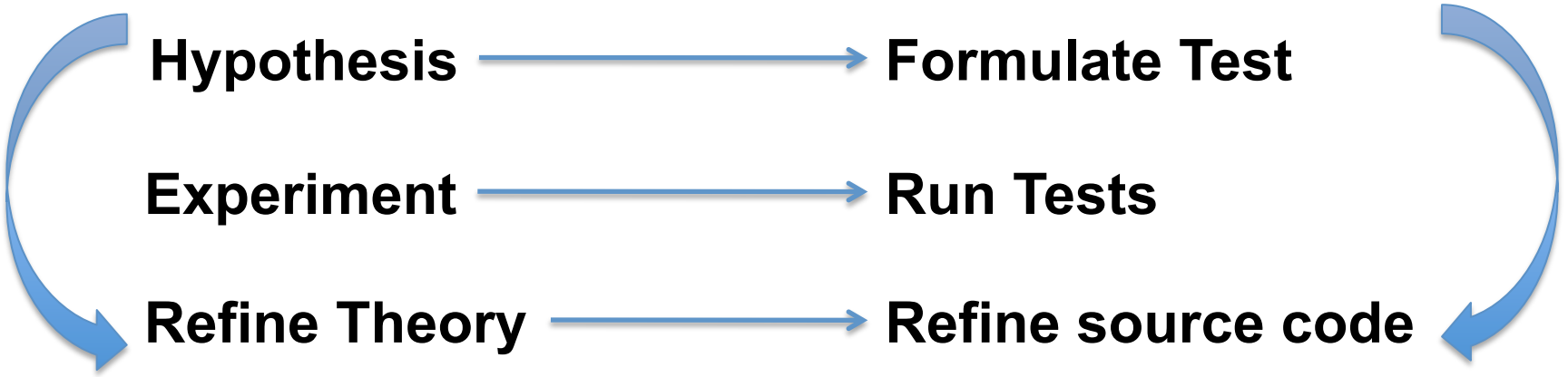
- Print:

```
 print*, "loss of mass = ", deltaMass
```

- Visual inspection / acceptance threshold:



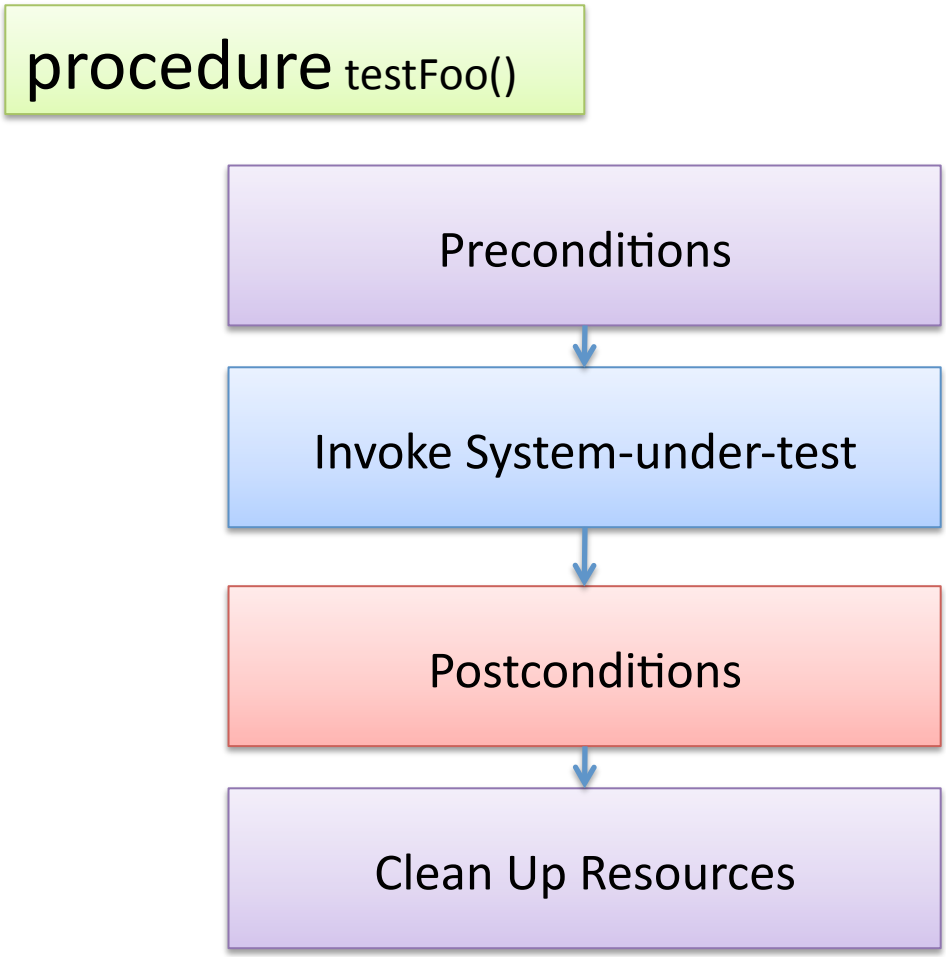
TDD and the Scientific Method



Properties of Good Tests

- Isolated
  - Failure indicates which part of application
- Orthogonal
  - Any bug only triggers small set of tests
- Independent
  - Run order does not matter
  - Corollary – cannot terminate execution
- Small
  - Execute quickly; small drain on resources
- Automated and repeatable

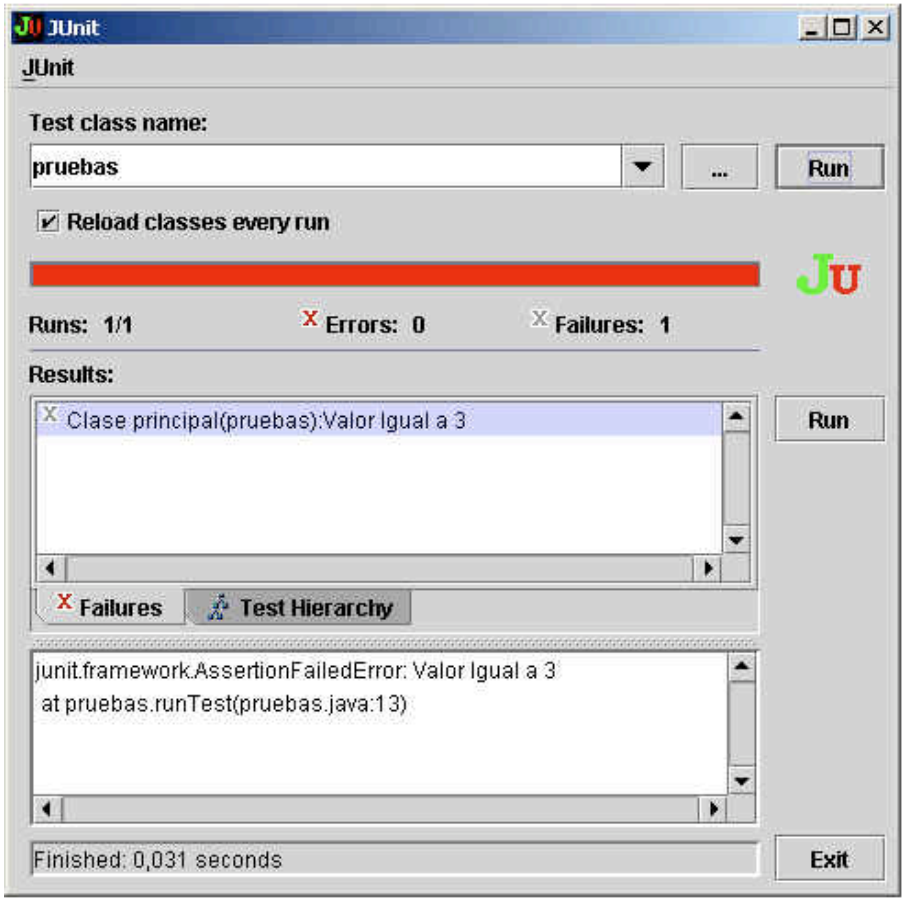
Anatomy of a Test Procedure

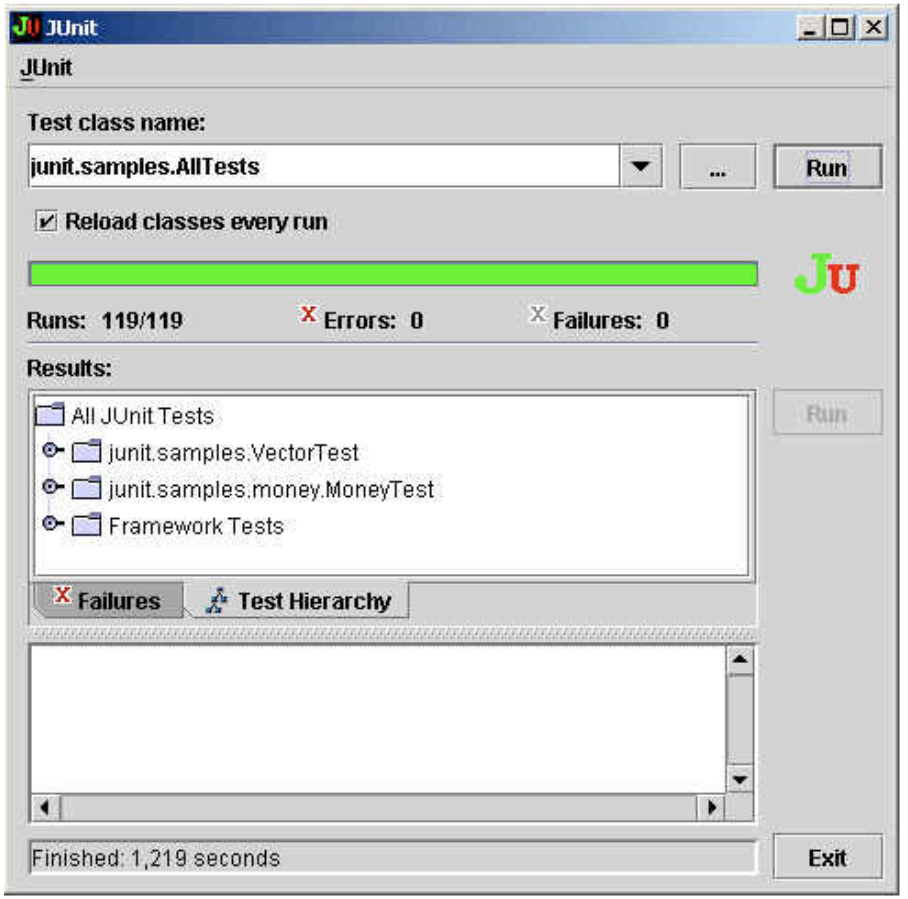


Testing Frameworks

- Provide infrastructure to *radically* simplify:
  - Creating test routines (Test cases)
  - Running collections of tests (Test suites)
  - Summarizing results
- Key feature is collection of "assert" methods
  - Used to express expected results
  - E.g. `assertEqual(120, factorial(5))`
- Generally specific to programming language (xUnit)
  - Java (JUnit), Python (pyUnit), C++ (cxxUnit, cppUnit)

JUnit - Eclipse



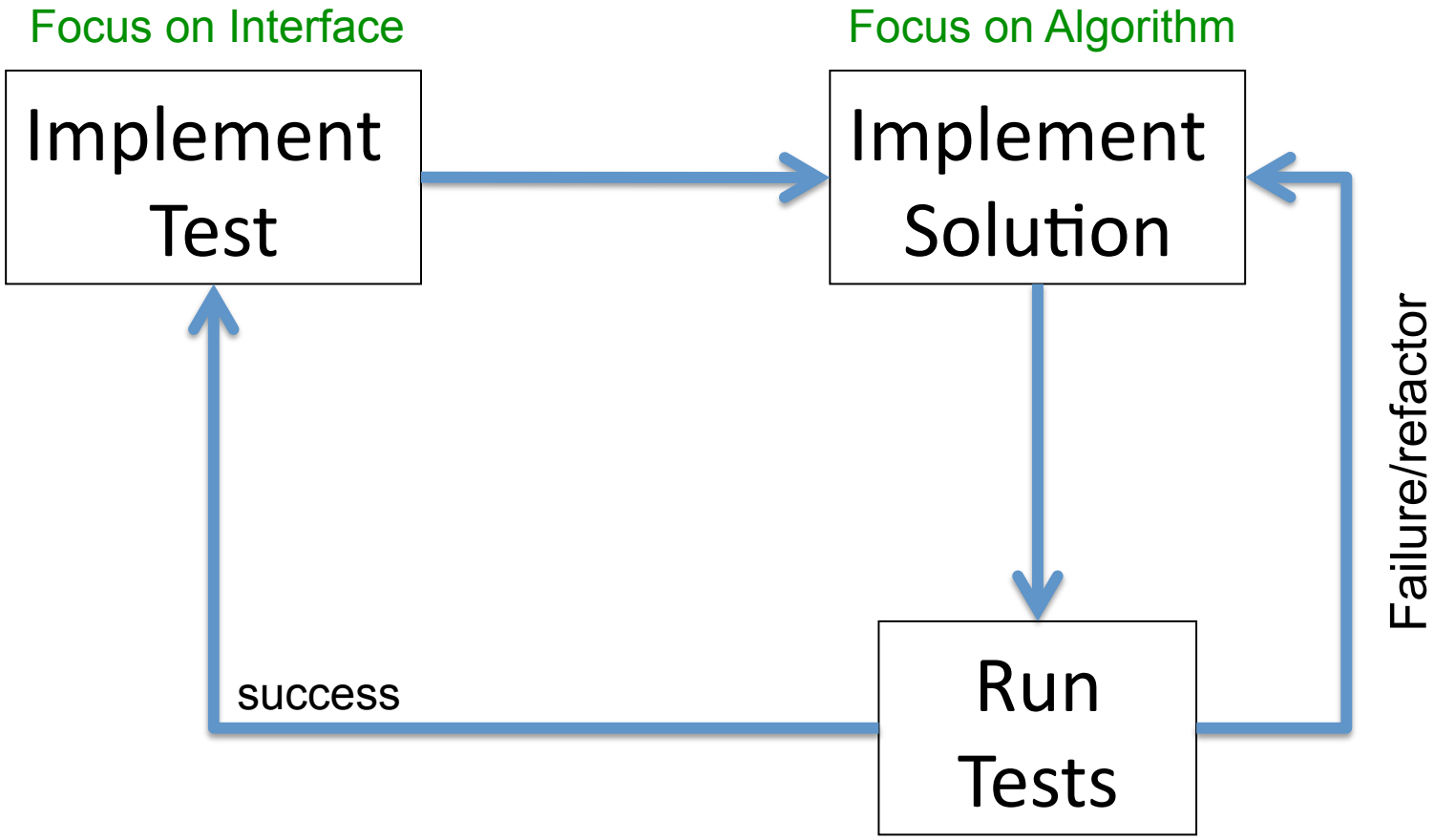


Test Driven Development

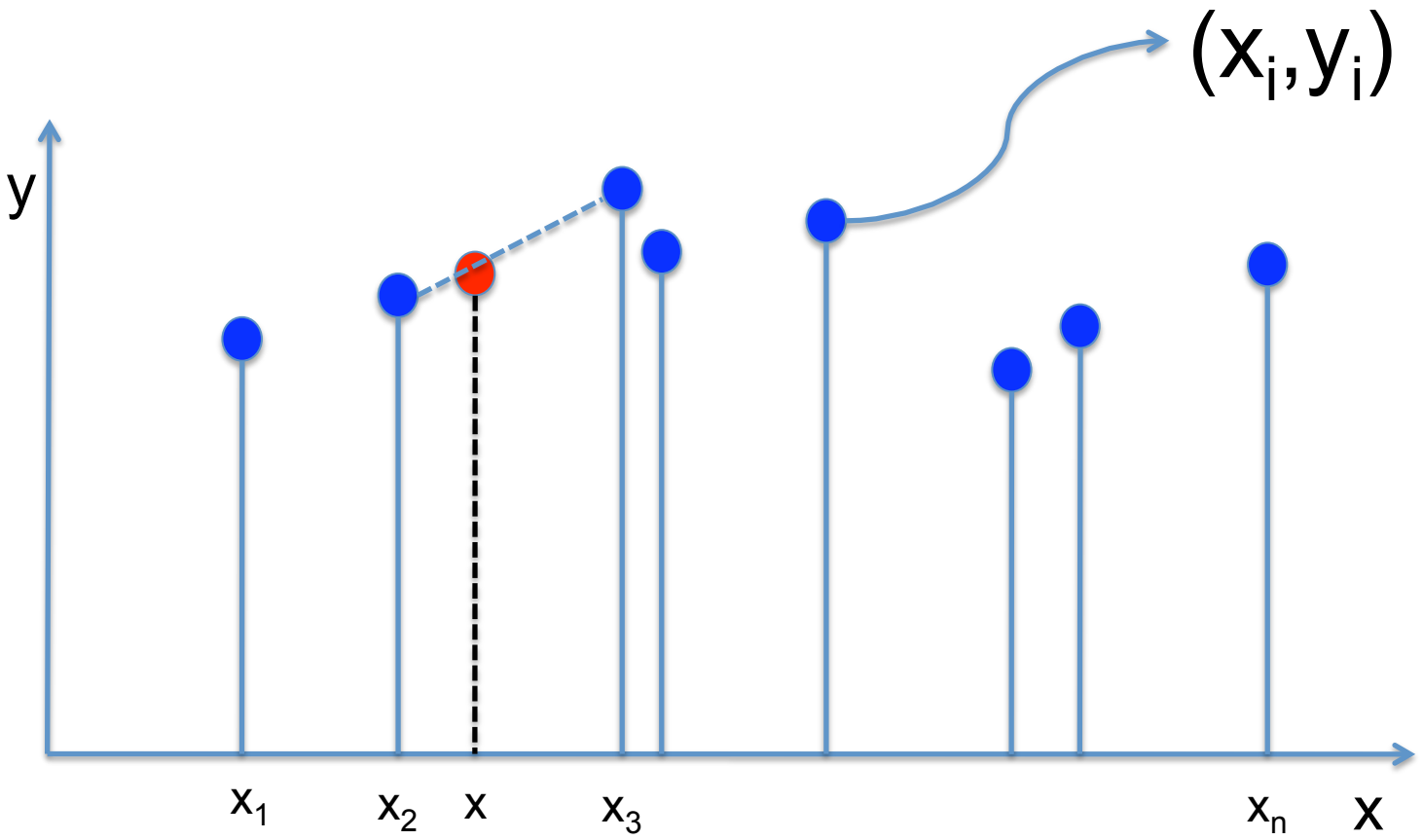
The Short Version

- Use tests to ***drive*** development
  1. Write a test (make it fail)
  2. Implement code to pass test
  3. Simplify/refactor/eliminate redundancy
  4. Rinse-and-repeat

The TDD Cycle



Example: Linear Interpolation



Potential Tests

- Bracketing: Find i such that $x_i <= x < x_{i+1}$
- Computing node weights:

  $w_a = (x_{i+1}-x)/(x_{i+1}-x_i)$

  $w_b = 1 - w_a$

- Weighted Sum: $y = w_a\, y_i + w_b\, y_{i+1}$

Possible Bracketing tests

- X_i ={1,2,3}; x = 1.5; return: index = 1
- X_i ={1,2,3}; x = 2.5; return: index = 2
- X_i ={1,2,3}; x = 2.0; return: index = 2 ???
- X_i ={1,2,3}; x = 1.0; return: index = 1
- X_i ={1,2,3}; x = 3.0; return: index = 2 ???
- X_i ={1,2,3}; x = 0.5; out-of-bounds error?
- X_i ={3,2,1}; x = 1.5; inverted-order error?

Bracketing: Test 1

- X_i ={1,2,3}; x = 1.5; return: index = 1

```
subroutine testBracket1()
   nodes = [1.,2.,3.]
   index = getBracket(nodes, 1.5)
   call assertEqual(1, index)
end subroutine
```

```
function getBracket(nodes, x)
   return 1
end function
```

Bracketing: Test 2

- X_i ={1,2,3}; x = 1.5; return: index = 2

```
subroutine testBracket2()
   nodes = [1.,2.,3.]
   index = getBracket(nodes, 2.5)
   call assertEqual(2, index)
end subroutine testBracket2
```

```
function getBracket(nodes, x)
   if (x > nodes(2)) then
      return 2
   else
      return 1
   end if
end function
```

```
function getBracket(nodes, x)
   do i = 1, size(nodes) – 1
      if (nodes(i+1) > x) return i
   end do
end function
```

```
end function
```

Tests for Weights

- X_i ={1,2}; x = 1.0; w_1 = 1.0
- X_i ={1,2}; x = 2.0; w_1 = 0.0
- X_i ={1,2}; x = 1.5; w_1 = 0.5
- X_i ={1,3}; x = 1.5; w_1 = 0.75
- X_i ={1,1}; x = 1.0; duplicate-node error

Weights: Test 1

- $X_i = \{1,2\}$; x = 1.0; $w_1 = 1.0$

```
subroutine testWeight1()
   [a,b] = [1,2]
   weight = computeWeight(a, b, 1.0)
   call assertEqual(1.0, weight)
end subroutine testWeight1
```

Duplication

```
subroutine computeWeight(a, b, x)
   return 1.0
end subroutine computeWeight
```

Interpolation: Test 1

- Constant Y

```
subroutine testInterpolate1()
   nodes = [[1,1],[2,1],[4,1]]
   y = interpolate(nodes, 3.0)
   call assertEqual(1.0, y)
end subroutine testInterpolate1
```

```
function interpolate(nodes, x)
   y = 1
end function interpolate
```

Interpolation: Test 2

- {(1,1),(2,3),(4,1)}; x = 3. => y(x) = 2

```
subroutine testInterpolate1()
   nodes = [[1,1],[2,3],[4,1]]
   y = interpolate(nodes, 3.0)
   assertEqual(2.0, y, epsilon)
end subroutine testInterpolate1
```

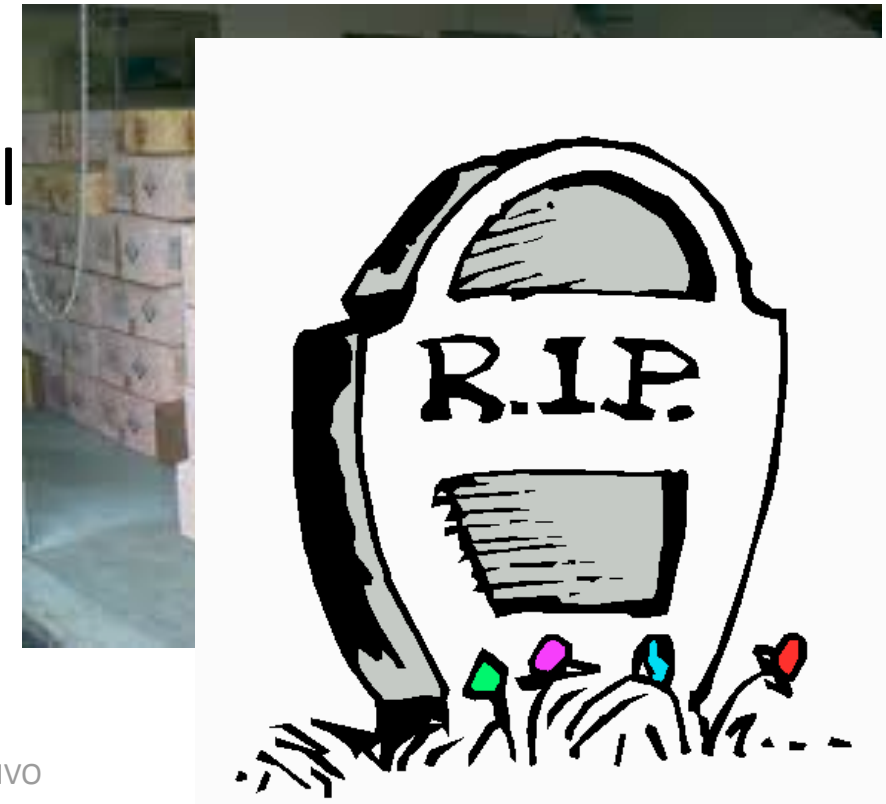
```
function interpolate(nodes, x)
   i = getBracket(nodes%xCoord, x)
   a = computeWeight(xc(i), xc(i+1), x)
   b = 1 – a
   return a*nodes(i)%yCoord + b*nodes(i+1)%yCoord
end function interpolate
```

TDD Best Practices

- Small steps - each iteration < 10 minutes
  - Starting over is cheap
  - Compilation speed sets lower bound (use –O0)
- Isolated, orthogonal, small, clear tests
- Extremely fast tests – need to run 1000's
  - Each test < 0.001 seconds
  - Don't need ¼ degree resolution to test *software*
- *Ruthless* refactoring
- Check that each test initially ***fails***

Benefits of TDD

- High software reliability
- Excellent test coverage
- Always ready-to-ship
- Tests act as maintainabl documentation
- Tests do not decay
- Debugging is rare



Benefits of TDD (cont'd)

- Reduced stress / improved confidence
- Productivity
- Predictable schedule
- High quality implementation
  - Test design requires focus
  - Testable code forces simple orthogonal interfaces
- Porting

Anecdotal Testimony

- Many professional SE's are initially skeptical
  - High percentage refuse to go back to "the old way" after only a few days of exposure.
- Projects that are able to drop bug tracking.
- Can be difficult to sell to management
  - "What? ***More*** lines of code?"

Not a Panacea

- Requires training, practice, and discipline
- Need strong tools (framework + refactoring)
- Does not invent new algorithms (e.g. FFT)
  - No such thing as magic
- Maintaining tests can be difficult during a major re-engineering effort.
  - But isn't the alternative is even worse?!!

TDD and Scientific Computing

Obstacles

- Difficult to apply to legacy software
- Developers are scientists; not SE's
- Limitations of Fortran
  - Weak development tools (but improving)
  - Not OO (impacts certain kinds of testing)
  - Lack of literature/training materials
- Need support for MPI, multi-dim arrays, etc.
- Numerical algorithms/parameterizations
  - Small number of analytic solutions
  - Specifying accuracy of floating-point results

TDD Experience in SIVO

- Software projects:
  - pFUnit, NED, DYNAMO, SMVGEAR,
    GTRAJ (C++), Sensor Web (Java/python), Snowfake
- Observations
  - Ratio of test code to source code is about 1:1
  - Works very well for *infrastructure*
  - Demonstrable improvements in quality
  - Learning curve
    - 1-2 days for technique
    - Weeks/months to wean from old habits

pFUnit

Parallel Fortran Unit Testing Framework

- Developed in SIVO using TDD (Clune and Womack)
- Supports testing of MPI-based applications
- Extensive support for floating-point and multi-dimensional arrays
- Available via NASA open-source license:
  http://sourceforge.net/projects/pfunit
- Possibly arrange a hands-on tutorial:
  - Contact Carlos Cruz if interested: Carlos.A.Cruz@nasa.gov

References

- pFUnit: http://sourceforge.net/projects/pfunit/
  - Tutorial materials
    https://modelingguru.nasa.gov/docs/DOC-1982
    https://modelingguru.nasa.gov/docs/DOC-1983
    https://modelingguru.nasa.gov/docs/DOC-1984
- TDD Blog:
  https://modelingguru.nasa.gov/blogs/modelingwithtdd
- *Test-Driven Development: By Example,* Kent Beck
- *Refactoring: Improving the Design of Existing Code,* Martin Fowler
- Junit, http://junit.sourceforge.net/